CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2017; 29: e3786
Published online 24 February 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3786

SPECIAL ISSUE PAPER

Optimizing hadoop parameter settings with gene expression programming guided PSO

Mukhtaj Khan[1,2,*,†], Zhengwen Huang[1], Maozhen Li[1,3], Gareth A. Taylor[1] and Mushtaq Khan[4]

[1]*Department of Electronic and Computer Engineering, Brunel University London, Uxbridge,UB8 3PH, UK*
[2]*Department of Computer Science, Abdul Wali Khan University Mardan, Mardan, Pakistan*
[3]*The Key Laboratory of Embedded Systems and Service Computing, Tongji University, Shanghai, China*
[4]*Department of Information Technology, Pakistan Ordnance Factory, Punjab, Pakistan*

SUMMARY

Hadoop MapReduce has become a major computing technology in support of big data analytics. The Hadoop framework has over 190 configuration parameters, and some of them can have a significant effect on the performance of a Hadoop job. Manually tuning the optimum or near optimum values of these parameters is a challenging task and also a time consuming process. This paper optimizes the performance of Hadoop by automatically tuning its configuration parameter settings. The proposed work first employs gene expression programming technique to build an objective function based on historical job running records, which represents a correlation among the Hadoop configuration parameters. It then employs particle swarm optimization technique, which makes use of the objective function to search for optimal or near optimal parameter settings. Experimental results show that the proposed work enhances the performance of Hadoop significantly compared with the default settings. Moreover, it outperforms both rule-of-thumb settings and the Starfish model in Hadoop performance optimization. © 2016 The Authors. *Concurrency and Computation: Practice and Experience* Published by John Wiley & Sons Ltd.

Received 30 October 2015; Accepted 28 December 2015

KEY WORDS: hadoop; mapreduce; big data analytics; gene expression programming; particle swarm optimization

1. INTRODUCTION

Many organizations are continuously collecting massive amounts of datasets from various sources such as the World Wide Web, sensor networks, and social networks. The ability to perform scalable and timely analytics on these unstructured datasets is a high priority for many enterprises. It has become difficult for traditional database systems to process these continuously growing datasets. Hadoop MapReduce has become a major computing technology in support of big data analytics [1, 2]. Hadoop has received a wide uptake from the community because of its remarkable features such as high scalability, fault-tolerance, and data parallelization. It automatically distributes data and parallelizes computation across a cluster of computer nodes [3–7].

*Correspondence to: Khan, Mukhtaj, Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, UB8 3PH, UK.
[†]E-mail: Mukhtaj.Khan@brunel.ac.uk

Despite these remarkable features, Hadoop is a large and complex framework, which has a number of components that interact with each other across multiple computer nodes. The performance of a Hadoop job is sensitive to each component of the Hadoop framework including the underlying hardware, network infrastructure, and Hadoop configuration parameters, which are over 190. Recent researches show that the parameter settings of the Hadoop framework play a critical role in the performance of Hadoop. A small change in the configuration parameter settings can have a significant impact on the performance of a Hadoop job [8]. Manually tuning the optimum or near optimum values of these parameters is a challenging task and also a time consuming process. In addition, the Hadoop framework has a black box like feature, which makes it extremely difficult to find a mathematical model or an objective function, which represents a correlation among the parameters. The large parameter space together with the complex correlations among the configuration parameters further increases the complexity of a manual tuning process. Therefore, an effective and automatic approach to tuning Hadoop parameters has become a necessity.

A number of research works have been proposed to automatically tune Hadoop parameter settings. The rule-of-thumb (ROT) proposed by industrial professionals [9–11] is just a common practice to tune Hadoop parameter settings. The Starfish optimizer [12, 13] optimizes the performance of a Hadoop job based on the job profile and a cost model [14]. The job profile is collected at a fine granularity with detailed information. However, collecting the detailed execution profile of a job incurs a high overhead, which overestimates the values for some configuration parameters. Moreover, the Starfish optimizer divides the search space into subspaces in the optimization process, which ignores the correlations among the configuration parameters. PPABS [15] automatically tunes Hadoop parameter settings based on the executed job profiles. Profiling and Performance Analysis-based System (PPABS) employs *K-means++* to classify the jobs into equivalent classes. It applies simulated annealing to search for optimum parameter values and implements a pattern recognition technique to determine the class that a new job belongs to. However, PPABS is unable to tune the parameter settings for a new job, which does not belong to any of the pre-classified classes. Gunther, a search based system proposed in [16] automatically searches for optimum parameter values for the configuration parameters using genetic algorithm. One critical limitation of Gunther is that it does not have a fitness function in the implemented genetic algorithm. Gunther evaluates the fitness of a set of parameter values by running a Hadoop job physically, which is a time consuming process. Panacea [17] optimizes Hadoop applications based on a process of tuning the configuration parameter settings. Similar to Starfish, Panacea also divides the search space into subspaces and then searches for optimal values within pre-defined ranges. The work presented in [18] proposes a performance evaluation model, which focuses on the impact of the Hadoop configuration settings from the aspects of hardware, software, and network.

Tuning the configuration parameters of Hadoop requires the knowledge of the internal dynamics of the Hadoop framework and the inter-dependencies among its configuration parameters. This is because the value of one parameter can have a significant impact on the other parameters. It should be pointed out that none of the aforementioned works considers the inter-dependencies among Hadoop configuration parameters. In this paper, we optimize the performance of Hadoop by automatically tuning its configuration parameter settings. The major contributions of this paper are as follows:

- Based on the running records of Hadoop jobs, which can be either CPU intensive or IO intensive, we employ gene expression programming (GEP) technique to build an objective function, which represents a correlation among the Hadoop configuration parameters. To the best of our knowledge, this is the first work that mathematically describes the inter-dependencies among the Hadoop configuration parameters when tuning the performance of Hadoop.
- For the purpose of configuration parameter optimization, particle swarm optimization (PSO) [19, 20] is employed that makes use of the GEP constructed objective function to search for a set of optimal or near optimal values of the configuration parameters. Unlike other optimization works that divide the search space into subspaces, the implemented PSO considers the whole search space in the optimization process in order to maintain the inter-dependencies among the configuration parameters.

DOI: 10.1002/cpe

To evaluate the performance of the proposed work, we run two typical Hadoop MapReduce applications, that is, WordCount and Sort, which are CPU and Input/Output (I/O) intensive, respectively. The performance of the proposed work is initially evaluated on an experimental Hadoop cluster configured with eight Virtual Machines (VMs) and subsequently on another Hadoop cluster configured with 16 VMs. The experimental results show that the proposed work enhances the performance of Hadoop by on average 67% on the WordCount application and 46% on the Sort application, respectively, compared with its default settings. The proposed work also outperforms both ROT and the Starfish model in Hadoop performance optimization.

The reminder of paper is organized as follows. Section 2 introduces a set of core configuration parameters of Hadoop considered in this work. Section 3 presents the design and implementation of GEP in generating an objective function, which represents a correlation of the Hadoop parameters. The implementation of the PSO for optimization of parameter settings is presented in Section 4. Section 5 evaluates the performance of the proposed work on two experimental Hadoop clusters. Section 6 discusses a number of related works. Section 7 concludes the paper and points out some further work.

2. HADOOP CORE PARAMETERS

The Hadoop framework has more than 190 tunable configuration parameters that allow users to manage the flow of a Hadoop job in different phases during the execution process. Some of them are core parameters and have a significant impact on the performance of a Hadoop job [12,16]. The core parameters are briefly presented in Table I.

2.1. io.Sort.Factor

This parameter determines the number of files (streams) to be merged during the sorting process of map tasks. The default value is 10, but increasing its value improves the utilization of the physical memory and reduces the overhead in IO operations.

2.2. io.Sort.mb

During a job execution, the output of a map task is not directly written into the hard disk but is written into an in-memory buffer, which is assigned to each map task. The size of the in-memory buffer is specified through the *io.sort.mb* parameter. The default value of this parameter is 100 MB. The recommended value for this parameter is between 30% and 40% of the

Table I. Hadoop core configuration parameters.

Configuration parameters	Default values	Brief descriptions
io.sort.factor	10	The number of streams that can be merged while sorting.
io.sort.mb	100	The size of the in-memory buffer assigned to each task.
io.sort.spill.percent	0.8	A threshold which determines when to start the spill process, transferring the in-memory data into the hard disk.
mapred.reduce.tasks	1	The number of reduce task(s) configured for a Hadoop job.
mapreduce.tasktracker.map.tasks.maximum	2	The number of map slots configured on each worker node.
mapreduce.tasktracker.reduce.tasks.maximum	2	The number of reduce slots configured on each worker node.
mapred.child.java.opts	200	The maximum size of the physical memory of JVM for each task.
mapreduce.reduce.shuffle.input.buffer.percent	0.70	The amount of memory in percentage assigned to a reducer to store map results during the shuffle process.
mapred.reduce.parallel.copies	5	The number of parallel data transfers running in the reduce phase.
mapred.compress.map.output	False	Compression of map task outputs.
mapred.output.compress	False	Compression of reduce task outputs.

DOI: 10.1002/cpe

Java_Opts value and should be larger than the output size of a map task, which minimizes the number of spill records [11].

2.3. io.Sort.Spill.Percent

The default value of this parameter is 0.8 (80%). When an in-memory buffer is filled up to 80%, the data of the in-memory buffer (*io.sort.mb*) should be spilled into the hard disk. It is recommended that the value of *io.sort.spill.percent* should not be less than 0.50.

2.4. Mapred.Reduce.Tasks

This parameter can have a significant impact on the performance of a Hadoop job [21]. The default value is 1. The optimum value of this parameter is mainly dependent on the size of an input dataset and the number of reduce slots configured in a Hadoop cluster. Setting a small number of reduce tasks for a job decreases the overhead in setting up tasks on a small input dataset, while setting a large number of reduce tasks improves the hard disk IO utilization on a large input dataset. The recommended number of reduce tasks is 90% of the total number of reduce slots configured in a cluster [8].

2.5. Mapreduce.Tasktracker.map.Tasks.Maximum, mapreduce.Tasktracker.Reduce.Tasks.Maximum

These parameters define the number of the map and reduce tasks that can be executed simultaneously on each cluster node. Increasing the values of these parameters increases the utilization of CPUs and physical memory of the cluster node, which can improve the performance of a Hadoop job. The optimum values of these parameters are dependent on the number of CPUs, the number of cores in each CPU, multi-threading capability, and the computational complexity of a job. The recommended values for these parameters are the number of CPU cores minus 1 as long as the cluster node has sufficient physical memory [9–11]. One CPU is reserved for other services in Hadoop such as DataNode and TaskTracker.

2.6. Mapred.Child.Java.Opts

This is a memory related parameter and the main candidate for JVM tuning. The default value is *–Xmx200m,* which gives at most 200 MB physical memory to each child task. Increasing the value of *Java_Opt* reduces spill operations to output map results into the hard disk, which can improve the performance of a job. By default, each work node utilizes 2.8GB physical memory [11]. The worker node assigns 400 MB to the map phase (i.e., two map slots), 400 MB to the reduce phase (i.e., two reduce slots) and 1000 MB to each DataNode and TaskTracker that run on the worker node.

2.7. Mapred.Compress.map.Output, mapred.Output.Compress

These two parameters are related to the hard disk IO and network data transfer operations. *Boolean* values are used to determine whether or not the map output and the reduce output need to be compressed. Enabling the compression of the map and reduce outputs for a job can speed up the hard disk IO and minimize the overhead in data shuffling across the network.

3. MINING HADOOP PARAMETER CORRELATIONS WITH GEP

Gene expression programming [22] is a new type of evolutionary algorithm [23]. It is developed based on a similar idea to genetic algorithms [24] and genetic programming [25]. Using a special format of the solution representation structure, GEP overcomes some limitations of both genetic algorithms and genetic programming. GEP brings a significant improvement on problems such as combinatorial optimization, classification, time series prediction, parametric regression, and symbolic regression.

DOI: 10.1002/cpe

GEP has been applied to a variety of domains such as data analysis in high energy physics, traffic engineering for IP networks, designing electronic circuits, and evolving classification rules. It has also been applied to data mining field especially for the investigation of an internal correlation among the involved parameters.

Gene expression programming uses a chromosome and expression tree combined structure [22] to represent a targeted problem being investigated. The factors of the targeted problem are encoded into a linear chromosome format together with some potential functions, which can be used to describe a correlation of the factors. Each chromosome generates an expression tree, and the chromosomes containing these factors are evolved during the evolutionary process.

3.1. Gene expression programming design

The execution time of a Hadoop job can be expressed in Eq.(1) where $x_0, x_1, \ldots, x_n$ are the configuration parameters of Hadoop.

$$ExecutionTime = f(x_0, , x_1, \ldots, , x_n) \tag{1}$$

In this work, we consider 10 core parameters of Hadoop as listed in Table II. Based on the data types of these Hadoop configuration parameters, the mathematic functions shown in Table III are used in GEP. A correlation of the Hadoop parameters can be represented by a combination of these mathematical functions. Figure 1 shows an example of mining a correlation of two parameters (x_0 and x_1), which is conducted in the following steps in GEP:

- Based on the data types of x_0 and x_1, find a mathematical function that has the same input data type as either x_0 or x_1 and has two input parameters.
- Calculate the estimated execution time of the selected mathematical function using the parameter setting samples.
- Find the best mathematical function between x_0 and x_1 that produces the closest estimated execution time to the actual execution time. In this case, the *Plus* function is selected.

Similarly, a correlation of $x_0, x_1, \ldots, x_n$ can be mined using the GEP method. The chromosome and expression tree structure of GEP is used to hold the parameters and mathematical functions. A combination of mathematical functions, which takes $x_0, x_1, \ldots, x_n$ as inputs, is encoded into a linear chromosome, which is maintained and developed during the evolution process. Meanwhile, the expression tree generated from the linear chromosome produces a form of $f(x_0, x_1, \ldots, x_n)$ based on which an estimated execution time is computed and compared with the actual execution time. A final form of $f(x_0, x_1, \ldots, x_n)$ will be produced at the end of the evolution process whose estimated execution time is the closest to the actual execution time.

In GEP, a chromosome can consist of one or more genes. For simplicity in computation, each chromosome has only one gene in this work. A gene is composed of a head and a tail. The

Table II. Hadoop core configuration parameters in gene expression programming.

Gene expression programming variables	Hadoop configuration parameters	Data types
x_0	io.sort.factor	integer
x_1	io.sort.mb	integer
x_2	io.sort.spill.percent	float
x_3	mapred.reduce.tasks	integer
x_4	mapreduce.tasktracker.map. tasks.maximum	integer
x_5	mapreduce.tasktracker.reduce.tasks.maximum	integer
x_6	mapred.child.java.opts	integer
x_7	mapreduce.reduce.shuffle.input.buffer.percent	float
x_8	mapred.reduce.parallel.copies	integer
x_9	input dataset size (GB)	integer

DOI: 10.1002/cpe

Table III. Mathematic functions used in gene expression programming.

Functions	Function Descriptions	Input Data Types
Plus	f(a,b) = a + b	integer or float
Minus	f(a,b) = a − b	integer or float
multiply	f(a,b) = a * b	integer or float
Divide	f(a,b) = a/b	integer or float
Sin	f(a) = sin(a)	integer or float
Cos	f(a) = cos(a)	integer or float
Tan	f(a) = tan(a)	integer or float
Acos	f(a) = acos(a)	integer or float
Asin	f(a) = asin(a)	integer or float
Atan	f(a) = atan(a)	integer or float
Exp	f(a) returns the exponential e^a	integer or float
Log	f(a) = log(a)	positive integer or float
log10	f(a) returns the (base-10) logarithm of a	positive integer or float
Pow	f(a,b) returns base a raised to the power exponent b	integer or float
Sqrt	f(a) = sqrt(x)	positive integer or float
Fmod	f(a,b) returns the floating-point remainder of a/b (rounded towards zero)	integer or float
pow10	f(a) returns base 10 raised to the power exponent a	integer or float
Inv	f(a) = 1/a	integer or float
Abs	f(a) returns absolute value of parameter a	integer
Neg	f(a) = −a;	integer or float

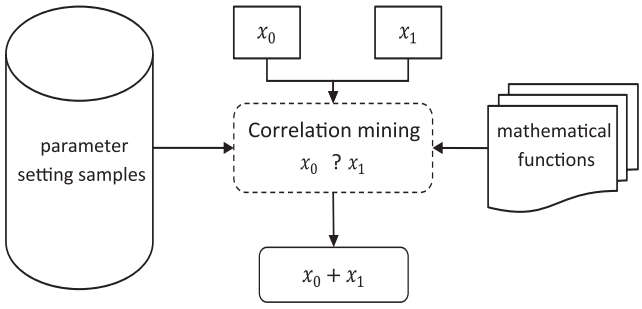

Figure 1. An examples of parameter correlation mining.

elements of the head are selected randomly from the set of Hadoop parameters (listed in Table II) and the set of mathematical functions (listed in Table III). However, the elements of the tail are selected only from the Hadoop parameter set. The length of a gene head is set to 20, which cover all the possible combinations of the mathematical functions. The length of a gene tail can be computed using Eq. (2).

$$Length(Gene_{Tail}) = Length(Gene_{Head}) \times (n-1) + 1 \tag{2}$$

where n is the number of input arguments of a mathematical function, which has the most number of input arguments among the functions. Figure 2 shows an example of a chromosome and expression tree structure taking into account five parameters - x_0, x_1, x_2, x_3, x_4.

In Figure 2, the size of the gene head is 4 and n is 2. Then the size of the gene tail is 5 based on Eq.(2). Four mathematical functions $(+, -, /, pow)$ are selected to represent a correlation of the parameters x_0, x_1, x_2, x_3, x_4. As a result, a form of $f(x_0, x_1, \ldots, x_n)$ is generated from the expression tree as illustrated in Eq. (3).

$$f(x_0, , x_1, , x_2, , x_3, , x_4) = (pow(x_3, , x_4) - x_0) + (x_1/x_2) \tag{3}$$

DOI: 10.1002/cpe

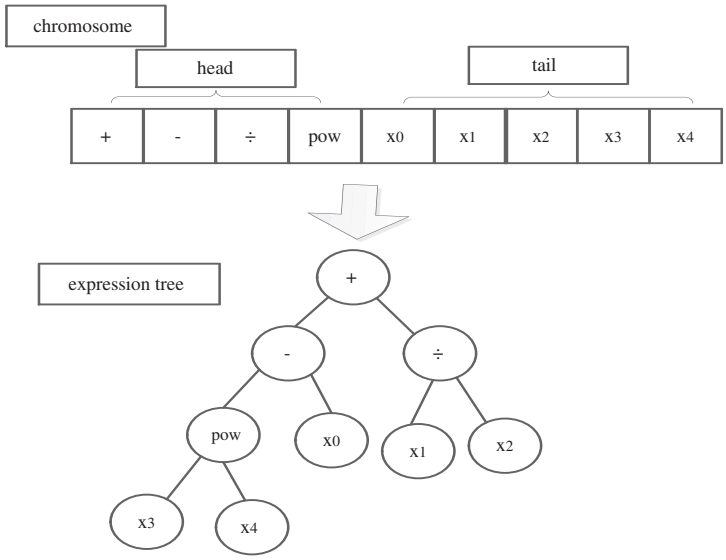

Figure 2. An example of chromosome and expression tree structure.

In the following section, we present how the GEP method evolves in mining a correlation among the Hadoop configuration parameters.

3.2. Gene expression programming implementation

Algorithm 1 shows the implementation of the GEP method. The input of the Algorithm 1 is a set of Hadoop job running samples, which are used as a training dataset. To build the training dataset, we conducted 320 experiments on a Hadoop cluster, which is presented in Section 5.1. We run two typical Hadoop applications (i.e., WordCount and Sort) to process an input dataset of different sizes ranging from 5GB to 15GB. For each experiment, we manually tuned the configuration parameter values and run the two applications three times each and took an average of the execution times. A small portion of the training dataset is presented in Table IV.

In Algorithm 1, Lines 1 to 5 initialize the first generation of 500 chromosomes, which represent 500 possible correlations among the Hadoop parameters. Lines 8 to 29 implement an evolution process in which a single loop represents a generation of the evolution process. For each chromosome, it is translated into an expression tree. Lines 11 to 17 calculate the fitness value of a chromosome. For each training sample, GEP produces an estimated execution time of a Hadoop job and makes a comparison with the actual execution time of the job. If the difference is less than a pre-defined bias window, the fitness value of the current chromosome will be increased by 1.

Table IV. Training data samples.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	Execution time (s)
20	70	0.70	4	2	2	150	0.70	5	5	539
40	80	0.81	16	3	1	200	0.80	8	5	518
50	75	0.73	14	3	2	210	0.85	4	5	510
120	65	0.85	8	1	1	150	0.75	5	5	540
200	73	0.82	8	3	3	260	0.79	8	10	898
200	66	0.85	12	2	2	160	0.75	5	10	857
200	70	0.81	8	2	1	180	0.73	5	10	877
100	100	0.66	16	2	2	200	0.70	5	15	1387

DOI: 10.1002/cpe

Input: A set of Hadoop job running samples;
Output: A correlation of the Hadoop parameters;

```
1:  FOR x = 1 TO size of population DO
2:    create chromosome(x) with the combination of mathematic function and parameter ;
3:      fitness value(x) = 0 ;
4:      x++;
5:  ENDFOR
6:  best chromosome = chromosome(1);
7:  best fitness value = 0;
8:  WHILE  i < termination generation number DO
9:    FOR  x = 1 TO size of population DO
10:         Translate chromosome(x) into expression tree(x);
11:         FOR y = 1 TO the number of training samples DO
12:                evaluate the estimated execution time for case(y)
13:           IF ABS(timeDiff) < bias window THEN
14:                fitness value(x)++;
15:           ENDIF
16:           y++;
17:       ENDFOR
16:          y++;
17:       ENDFOR
18:       IF fitness value(x) = the number of training samples THEN
19:                best chromosome = Chromosome(x) GOTO 29;
20:      ELSE IF fitness value(x) > best fitness value THEN
21:                best chromosome = Chromosome(x);
22:                best fitness value = fitness value(x) ;
23:       ENDIF
24:           Apply replication, selection and genetic modification on chromosome(x) proportionally;
25:          Use the modified chromosome(x) to overwrite the original one;
26:       x++;
27:    ENDFOR
28:    i++;
29:  ENDWHILE
30:  Return best chromosome
```

Algorithm 1: GEP implementation.

The size of the bias window is set to 50 s, which allows a maximum of 10% of the error space taking into account the actual execution time of a Hadoop job sample. Line 18 shows that the evolution process terminates in an ideal case when the fitness value is equal to the number of training samples. Otherwise, the evolution process continues, and the chromosome with the best fitness value will be kept as shown in Lines 20 to 23. At the end of each generation as shown in Lines 24 to 25, a genetic modification is applied to the current generation to generate variations of the chromosomes for the next generation.

We varied the number of generations from 20000 to 80000 in the GEP evolution process and found that the quality of a chromosome (the ratio of the fitness value to the number of training samples) was finally higher than 90%. As a result, we set 80000 as the number of generations. The genetic modification parameters were set using the classic values [22] as shown in Table V. After 80000 generations, GEP generates Eq. (4), which represents a correlation of the Hadoop parameters listed in Table II.

$$f(x_0, x_1, \ldots, x_9) = (x_7{*}x_6) + (sqrt(1/((log10(x_6) + mod(sqrt((x_0{*}x_8) + (x_3{*}x_1)), pow(x_5, (x_2 + x_1)))) + (x_6 + x_4))){*}(x_8 + x_9)) \tag{4}$$

4. HADOOP PARAMETER OPTIMIZATION WITH PSO

In this section, we employ PSO to optimize Hadoop parameter settings. We use Eq. (4) generated by the GEP method in Section 3.2 as an objective function in PSO optimization.

DOI: 10.1002/cpe

Table V. Gep parameter settings.

Genetic modification parameters of gene expression programming	Values (%)
one-point recombination rate	30
insertion sequence transposition rate	10
inversion rate	10
mutation rate	0.44

Particle swarm optimization is a kind of an evolutionary computational algorithm introduced by Eberhart and Kennedy in 1995. The algorithm is inspired by the social behaviors of bird flocking, fish schooling, and swarm theory [19, 20]. PSO has been successfully applied in a wide range of problem domains due to its rapid convergence process towards an optimum solution [26–30]. In PSO, particles can be considered as agents that fly through a multidimensional search space and record the best solution that they have discovered. Each particle of the swarm adjusts its path according to its own flying experience and also the flying experiences of its neighborhood particles in a multidimensional search space.

Let

- d be the number of dimensions of a search space. In this work, d is set to 9, which represents the 9 Hadoop configuration parameters listed in Table II.
- n be the total number of particles in a swarm.
- $X_{i,j}$ be the list of positions of the particle i, $X_{i,j}=(x_{i,1}, x_{i,2}, x_{i,3},..... x_{i,d})$, j is a dimension of the search space.
- $P_{i,j}$ be a list of the locally best positions of the particle i, $P_{i,j}=(p_{i,1}, p_{i,2}, p_{i,3},....., p_{i,d})$.
- $V_{i,j}$ be the velocity of the particle i, $V_{i,j}=(v_{i,1}, v_{i,2}, v_{i,3},...., v_{i,d})$.
- G be the list of the globally best positions of a swarm, $G=(g_1, g_2,....., g_d)$.

To implement the PSO algorithm, we first initialize the positions of the particles randomly within the bounds of the search space so that the search space is uniformly covered, while the velocities of the particles are initialized to zeros as suggested in [31]. Then the PSO algorithm updates the swarm by updating the velocity and position of each particle in every dimension using Eq. (5) and Eq. (6), respectively.

$$v_{i,j}^{t+1} = w \times v_{i,j}^{t} + c_1 \times r_1 \left(p_{i,j}^{t} - x_{i,j}^{t} \right) + c_2 \times r_2 \left(g_j^{t} - x_{i,j}^{t} \right) \tag{5}$$

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1} \tag{6}$$

where

- $r_1$and r_2 are cognitive and social randomization parameters, respectively. They have random values between 0 and 1.
- c_1 and c_2 are local and global weights, respectively. They are acceleration constants.
- w is an inertia weight that balances the global and local search capabilities [32].
- t is a relative time index.
- $v_{i,j}^{t+1}$is the velocity of the particle i at time step t + 1.
- $v_{i,j}^{t}$ is the velocity of the particle i at time step t.
- $p_{i,j}^{t}$ is the locally best position of the particle i at time step t.
- $x_{i,j}^{t}$ is the current position of the particle i at time step t.
- g_j^{t} is the globally best position visited by any particle at time step t .
- $x_{i,j}^{t+1}$is the new position of the particle i at time step t + 1.

In each iteration, the new position of a particle is evaluated using the objective function $f(x_0, x_1, \ldots, x_9)$. The locally best value is compared with the new fitness value and updated accordingly. Similarly, the globally best position is updated.

DOI: 10.1002/cpe

Input: The size of an input dataset in MB;
Output: A set of PSO recommended Hadoop parameter settings;

1. Initialization process;
2. **FOR** each particle i = 1 to the number of particles **DO**
3. **FOR** each dimension j = 1 to the number of dimensions **DO**
4. Initialize randomly the position $x_{i,j}$ within a search space ;
5. Initialize the velocity $v_{i,j} = 0$;
6. **ENDFOR**
7. fitness_value = $f(x_{i,j})$;
8. **IF** (fitness_value < locally_best_value) **THEN**
9. locally_best _value = fitness_value;
10. locally_best _position= $x_{i,j}$;
11. **ENDIF**
12. **IF** (fitness_value < globally_best_value) **THEN**
13. globally_best_value = fitness_value;
14. globally_best _position= $x_{i,j}$;
15. **ENDIF**
16. **ENDFOR**
17. **WHILE** (iteration < the number of iterations) **DO**
18. Compute the new velocity ($v_{i,j}$) using Eq.(5)
19. **IF** ($v_{i,j} < v_{j,\min}$) **THEN**
20. $v_{i,j} = v_{j,\min}$;
21. **ELSE IF** ($v_{i,j} > v_{j,\max}$) **THEN**
22. $v_{i,j} = v_{j,\max}$;
23. **ENDIF**
24. Compute the new position ($x_{i,j}$) using Eq.(6) ;
25. **IF** ($x_{i,j} < x_{j,\min}$) **THEN**
26. $x_{i,j} = x_{j,\min}$;
27. **ELSE IF** ($x_{i,j} > x_{j,\max}$) **THEN**
28. $x_{i,j} = x_{j,\max}$;
29. **ENDIF**
30. Evaluate the new position on fitness function f;
31. Update locally_best _position and globally_best _position ;
32. **ENDWHILE**
33. Output globally_best _position ;

Algorithm 2. PSO implementation.

In the PSO algorithm, clamping the velocity and position of a particle within a feasible search area is a challenging task. This task becomes even more complicated if the optimization problem has bounds. If the optimization problem has bounds then it is important to handle the particle positions along with the velocities flying out of the feasible area (i.e., out of boundary). In addition, it has been shown that as the number of problem parameters increases, the probability of the particles flying out of the feasible space increases dramatically [33], [34]. For this purpose, we employ the nearest method presented in [34] to handle bound violations.

To handle bound violations of a particle, we define $v_{j,\min}$ and $v_{j,\max,}$ which represent a lower bound and an upper bound of the velocity of the particle, respectively. Similarly, we define $x_{j,\min}$ and $x_{j,\max}$ representing a lower bound and an upper bound of the position of the particle, respectively. The values of the lower bound and the upper bound of the position of a particle are set according to the range of each Hadoop parameter listed in Table VI.

However, setting the values for the lower bound and the upper bound of the velocity of a particle is problem dependent, and the values can be found empirically. We set the value of $v_{j,\min}$ to (-10%) of $(x_{j,\max} - x_{j,\min})$ and the value of $v_{j,\max}$ to $(+10\%)$ of $(x_{j,\max} - x_{j,\min})$. Each particle moves in a search space following the upper and lower bounds of its position and velocity. If any particle is roaming then its velocity and position values are set back to the nearest bound values. Algorithm 2 shows the PSO implementation.

It is worth pointing out that sometimes, PSO can be trapped in a local optimum. This issue can be avoided by adjusting the inertia weight (w) factor used in Eq. (5). Instead of using a constant value for

DOI: 10.1002/cpe

Table VI. Hadoop parameter settings in particle swarm optimization.

Hadoop parameters	Values	Explanations
x_0	10 ~ 230	Empirically.
x_1	65 ~ 100	Based on the block size of an input dataset. We use 64 MB block size in Hadoop.
x_2	0.6 ~ 0 .85	Empirically.
x_3	1 ~ 16	Based on the total number of reduce slots configured in a Hadoop cluster.
x_4	1 ~ 3	Based on the specification of a worker node.
x_5	1 ~ 3	Based on the specification of a worker node.
x_6	180 ~ 6000	Based on the physical memory of a worker node and the x_1 value.
x_7	0.70 ~ 0.85	Empirically.
x_8	1 ~ 10	Empirically.
x_9	The size of an input dataset in MB	Specified by user.

w, we use a dynamic inertia weight that linearly decreases in every iteration to overcome the local optima problem [32]. The dynamic inertia weight can be computed using Eq. (7).

$$w = w_{\max} - (current_{iteration}/total_{iterations}) \times (w_{\max} - w_{\min}) \tag{7}$$

where $w_{\min} = 0$ and $w_{\max} = 1$.

5. PERFORMANCE EVALUATION

The performance of the GEP guided PSO optimization work was initially evaluated on an experimental Hadoop cluster using a single Intel Xeon server machine configured with eight VMs and subsequently on another Hadoop cluster using two Intel Xeon Server machines configured with 16 VMs. The intuition of using two Hadoop clusters was to intensively evaluate the performance of the proposed work by considering the network overhead across the two server machines. In this section, we first give a brief introduction to the experimental environments that were set up in the evaluation process and then present performance evaluation results.

5.1. Experimental set up

We set up a Hadoop cluster using one Intel Xeon server machine. The specification of the server is shown in Table VII. We installed Oracle Virtual Box and configured eight VMs on the server. Each VM was assigned with 4 CPU cores, 8GB RAM and 150GB hard disk storage. We installed Hadoop-1.2.1 and configured one VM as the Name Node and the remaining seven VMs as Data Nodes. The Name Node was also used as a Data Node. The data block size of the Hadoop Distributed File System (HDFS) was set to 64 MB, and the replication level of data block was set to 2.

The second experimental Hadoop cluster was set up on two Intel Xeon server machines. The specification of second server machine was the same as the first server machine as shown in Table VII. The total number of VMs in the second Hadoop cluster was 16. The Hadoop-1.2.1 version was installed, and we configured one VM as Name Node and the remaining 15 VMs as Data Nodes. The data block size of the HDFS was set to 64 MB, and the replication level of data block was set to 3. We run two typical Hadoop applications (i.e., WordCount and Sort) as Hadoop jobs. The TeraGen application of Hadoop was used to generate an input dataset of different sizes.

DOI: 10.1002/cpe

Table VII. Hadoop cluster set up.

Intel Xeon Server 1 and Server 2	CPU	40 cores
	Processor	2.27GHz
	Hard disk	2 TB
	Connectivity	100Mbps Ethernet LAN
	Memory	128GB
Software	Operating System	Ubuntu 12.04 TLS
	JDK	1.6
	Hadoop	1.2.1
	Oracle Virtual Box	4.2.8
	Starfish	0.3.0

5.2. The impact of hadoop parameters on performance

We run the WordCount application as a Hadoop job to evaluate the impacts of the configuration parameters listed in Table I on Hadoop performance. From Figure 3, it can be observed that the execution time of the job decreases with an increasing size of the *io.sort.mb* value. The larger size the parameter value has, the less operations will be incurred in writing the spill records to the hard disk leading to a less overhead in output.

The *io-sort-factor* parameter determines the number of data streams that can be merged in the sorting process. Initially, the execution time of the job goes down with an increasing value of the parameter as shown in Figure 4 that the value of 200 represents the best value of the parameter. Subsequently, the execution time goes up when the value of the parameter further increases. This is because that there is a tradeoff between the reduced overhead incurred in IO operations when the

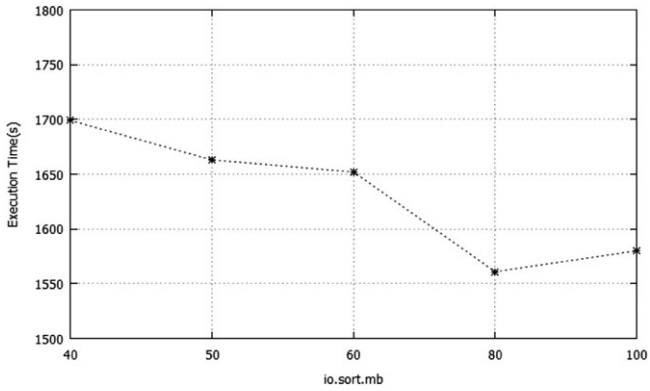

Figure 3. The impact of the *io.sort.mb* parameter.

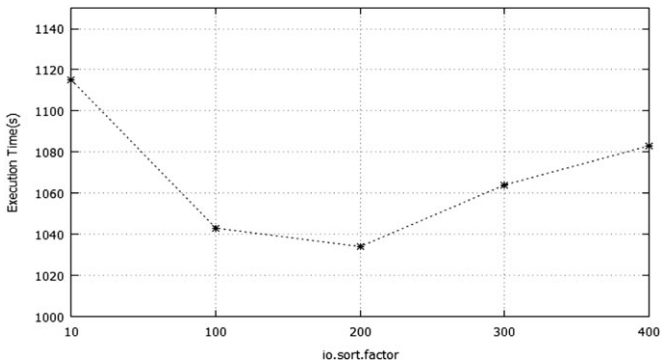

Figure 4. The impact of the *io-sort-factor* parameter.

DOI: 10.1002/cpe

value of the parameter increases and the added overhead incurred in merging the data streams.

Figure 5 shows the impact of the number of reduce tasks on the job performance. There is a tradeoff between the overhead incurred in setting up reduce tasks and the performance gain in utilizing resources. Initially increasing the number of reduce tasks better utilizes the available resources, which leads to a decreased execution time. However, a large number of reduce tasks incurs a high overhead in the setting up process, which leads to an increased execution time.

Increasing the number of map and reduce slots better utilizes available resources, which leads to a decreased execution time which can be observed in Figure 6 when the number of slots increases from 1 to 2. However, resources might be over utilized when the number of slots further increases, which slows down a job execution.

Increasing the value of *Java_opts* parameter utilizes more memory, which leads to a decreased execution time as shown in Figure 7. However, a large value of the parameter would over utilize the available memory space. In this case, the hard disk is used as a virtual memory, which slows down a job execution.

Figure 8 shows the impact of the compression parameter on the performance of a Hadoop job. The results generated by map tasks or reduce tasks can be compressed to reduce the overhead in IO operations and data transfer across network, which leads to a decreased execution time. It is worth noting that the performance gap between the case of using the *compression* feature and the case of using *uncompressing* feature gets large with an increasing size of the input data.

5.3. Particle swarm optimization set up

The parameters used in the PSO algorithm are presented in Table VIII. We set 20 for the particle swarm size and 100 for the number of iterations as suggested in the literature [35, 36]. The values of c_1 and c_2 were set to 1.4269 as proposed in [37], the value of w was set dynamically between 0 and 1, and the values of r_1 and r_2 were selected randomly between 0 and 1 in every iteration. The PSO algorithm

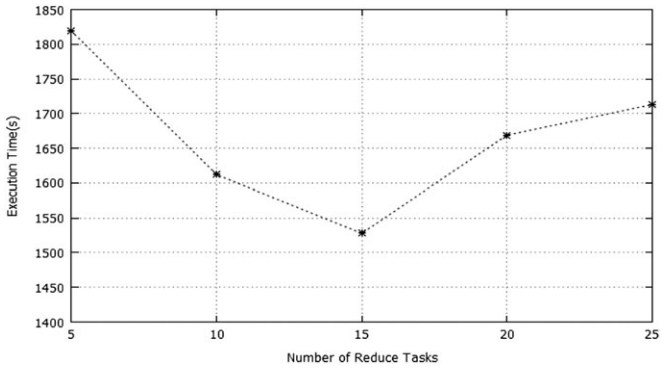

Figure 5. The impact of the number of reduce tasks.

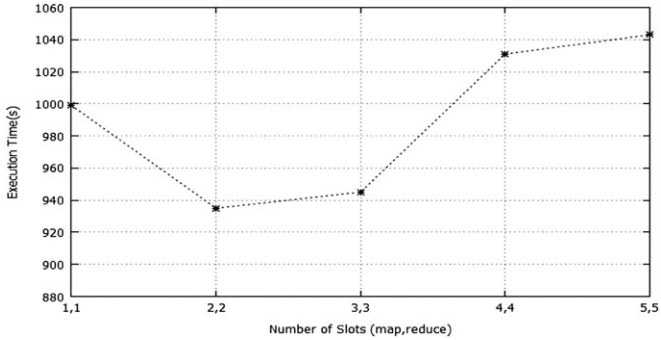

Figure 6. The impact of the number of map and reduce slots.

DOI: 10.1002/cpe

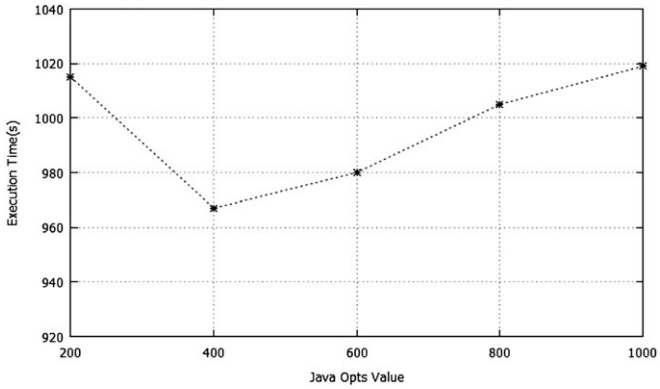

Figure 7. The impact of the *Java_opts* parameter.

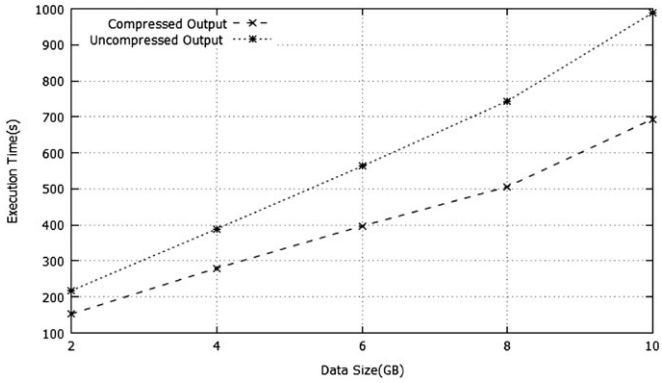

Figure 8. The impact of the compression parameter.

Table VIII. Particle swarm optimization parameter settings.

Swarm size	20
Number of iterations	100
c_1	1.4269
c_2	1.4269
w	[0,1]
r_1	Random [0,1]
r_2	Random [0,1]

processes *real number* values, while some of the Hadoop configuration parameters accept only *integer number* values (e.g., the number of map slots). We rounded the values of these PSO parameters to *integer* values. We set two configuration parameters which have a *Boolean* value (i.e., *mapred.compress.map.output* and *mapred.out.compress*) to *True*. This is because empirically, we found that the *True* values of these two parameters showed a significant improvement on the performance of a Hadoop job as shown in Figure 8.

Table IX presents the PSO recommended configuration parameter settings for a Hadoop job with an input dataset of varied sizes ranging from 5GB to 20GB.

5.4. Starfish Job profile

In order to collect a job profile for the Starfish optimizer, we first run both WordCount and Sort in the Starfish environment with *profiler* enabled. Both applications processed an input dataset of 5GB. Then the Starfish optimizer was invoked to generate configuration parameter settings. The recommended

DOI: 10.1002/cpe

Table IX. Particle swarm optimization recommended hadoop parameter settings on eight virtual machines.

Configuration parameters	Optimized values			
input dataset (GB)	5	10	15	20
io.sort.factor	230	228	213	155
io.sort.mb	100	93	100	91
io.sort.spill.percent	0.85	0.70	0.69	0.76
mapred.reduce.tasks	16	9	10	9
mapreduce.tasktracker.map.tasks.maximum	3	2	2	2
mapreduce.tasktracker.reduce.tasks.maximum	3	2	2	2
mapred.child.java.opts	280	335	420	553
mapreduce.reduce.shuffle.input.buffer.percent	7	7	7	7
mapred.reduce.parallel.copies	10	7	6	7
mapred.compress.map.output	True	True	True	True
mapred.output.compress	True	True	True	True

configuration parameter settings recommended by Starfish for both applications are presented in Table X and Table XI, respectively.

5.5. *Experimental results on hadoop performance*

In this section we compare the performance of the proposed work with that of Starfish, ROT, and the default configuration parameter settings in Hadoop optimization. Both WordCount and Sort applications were deployed on the Hadoop cluster with eight VMs to process an input dataset of four different sizes varying from 5GB to 20GB. We run both applications three times each using the PSO recommended parameter settings, and an average of the execution times was taken. The performance results of the two applications are shown in Figure 9 and Figure 10, respectively.

It can be observed that overall, the implemented PSO improves the performance of the WordCount application by an average of 67% in the four input data scenarios compared with the default Hadoop parameter settings, 28% compared with Starfish, and 26% compared with ROT. The improvement reaches a maximum of 71% when the input data size is 20GB. The performance improvement of the PSO optimization on the Sort application is on average 46% over the default Hadoop parameter settings, 16% over Starfish, and 37% over ROT. The improvement reaches a maximum of 65% when the input data size is 20GB.

It should be pointed out that the implemented PSO algorithm considers both the underlying hardware resources and the size of an input dataset and then recommends configuration parameter settings for both applications. The ROT work only considers the underlying hardware resources

Table X. Starfish recommend parameter settings for the WordCount applications on eight virtual machines.

Configuration parameters	Optimized values			
input dataset (GB)	5	10	15	20
io.sort.mb	117	129	128	120
io.sort.factor	35	50	17	76
mapred.reduce.tasks	32	128	176	192
shuffle.input.buffer.percentage	0.43	0.72	0.63	0.83
min.num.spills.for.combine	3	3	3	3
io.sort.spill.percent	0.86	0.85	0.79	.085
io.sort.record.percent	0.23	0.33	0.33	0.31
mapred.job.shuffle.merge.percent	0.86	0.85	0.83	0.69
mapred.inmem.merge.threshold	660	816	827	765
mapred.output.compress	True	True	True	True
mapred.compress.map.output	True	True	True	True
mapred.job.reduce.input.buffer.percent	0.42	0.43	0.60	0.77

DOI: 10.1002/cpe

Table XI. Starfish recommend parameter settings for the sort applications on eight virtual machines.

Configuration parameters	Optimized values			
input dataset (GB)	5	10	15	20
io.sort.mb	110	127	109	123
io.sort.factor	48	35	54	27
mapred.reduce.tasks	48	112	160	176
shuffle.input.buffer.percentage	0.76	0.66	0.63	0.88
io.sort.spill.percent	0.84	0.68	0.87	0.82
io.sort.record.percent	0.21	0.15	0.23	0.11
mapred.job.shuffle.merge.percent	0.77	0.88	0.89	0.76
mapred.inmem.merge.threshold	393	787	783	972
mapred.output.compress	True	True	True	True
mapred.compress.map.output	True	True	True	True
mapred.job.reduce.input.buffer.percent	0.65	0.63	0.52	0.79

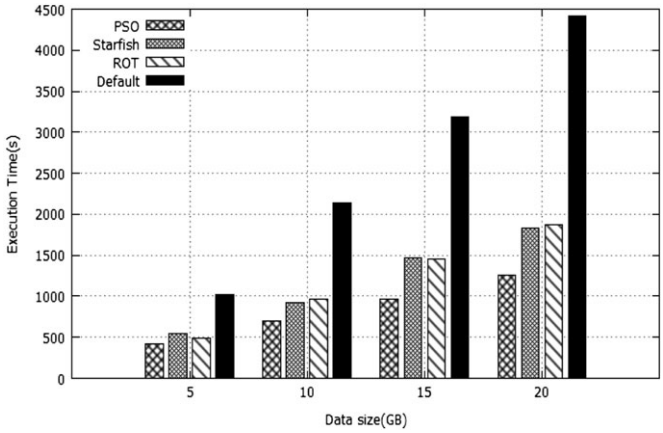

Figure 9. The performance of the WordCount application using eight virtual machines.

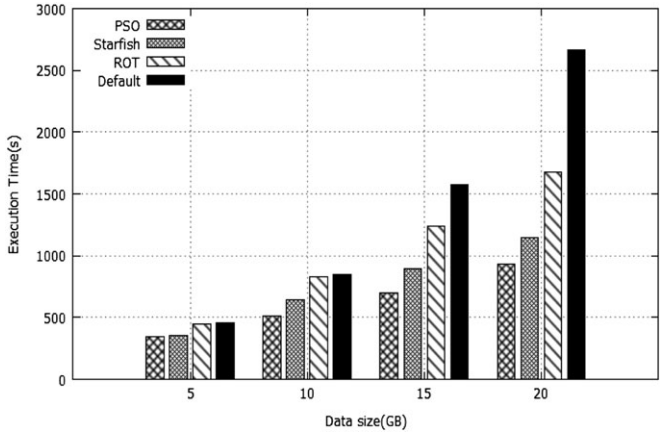

Figure 10. The performance of Sort application using eight virtual machines.

(i.e., CPUs and physical memory) and ignores the size of an input dataset. The Starfish model also considers both the underlying hardware resources and the size of an input dataset. However, Starfish overestimates the number of reduce tasks. For example, Starfish recommended 192 reduce tasks for the WordCount application and 176 reduce tasks for the Sort application on a 20GB dataset. A large number of reduce tasks improves hard disk utilization through task parallelization but generates a high overhead in setting up these reduce tasks in Hadoop. ROT ignores the input dataset size; therefore, the recommended parameter settings of ROT are the same for all the input datasets as

DOI: 10.1002/cpe

shown in Table XII. It is worth noting that ROT performs slightly better than Starfish on the WordCount application. This is because Starfish suggests a large number of reduce tasks, which generates a high overhead in setting up these reduce tasks, especially in the case of using a small input dataset (e.g., 5GB). Whereas ROT suggests a small number of reduce tasks, which are completed in a single wave generating a low overhead in setting up the reduce tasks. ROT estimates the number of reduce tasks based on the total number of reduce slots configured in the Hadoop cluster.

We have further evaluated the performance of the PSO optimization work on another Hadoop cluster configured with 16 VMs. From Figure 11 and Figure 12 it can be observed that the PSO work improves the performance of both applications on average by 65% and 86% compared with ROT and the default Hadoop settings, respectively. The improvement reaches a maximum of 87% when the input data size is 35GB on the WordCount application. The performance gains of the PSO work over the Starfish model on the WordCount application and the Sort application are on average 20% and 21%, respectively. It is worth noting that the Starfish model performs better than ROT in the case of using 16 VMs. In this case, a large dataset with a size varying from 25GB to 40GB was used. ROT recommends *False* for the *mapred.output.compress* parameter (as shown in Table XII). As a result, both applications took a long time in the reduce phase when writing the reduce task outputs into the hard disk. For example, the WordCount took 19 min to process the 40GB dataset in the map phase and 61 min in the reduce phase following the ROT recommended parameter settings. Whereas it took 13 minutes to process the same amount of data in the map phase and only 23 min in the reduce phase following the Starfish recommended parameter settings. This is because Starfish enabled the *mapred.output.compress* parameter, which reduces the overhead in writing the reduce task outputs into the hard disk.

Table XII. Rot recommended parameter settings on eight virtual machines.

Configuration Parameters	Optimized Values
io.sort.factor	25
io.sort.mb	250
io.sort.spill.percent	0.8
mapred.reduce.tasks	14
mapreduce.tasktracker.map.tasks.maximum	3
mapreduce.tasktracker.reduce.tasks.maximum	3
mapred.child.java.opts	600
input.buffer.percent	0.7
mapred.reduce.parallel.copies	20
mapred.compress.map.output	True
mapred.output.compress	False

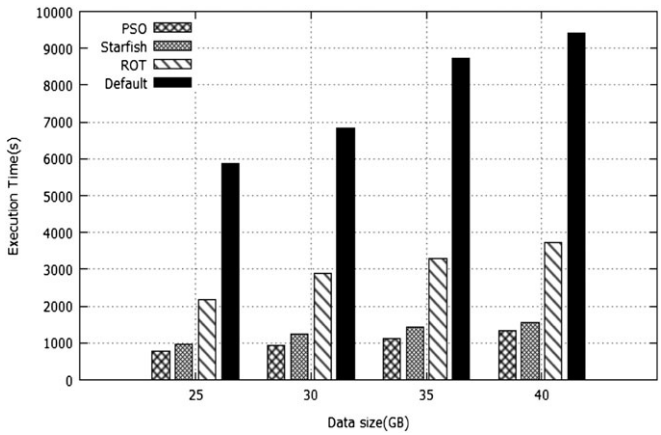

Figure 11. The performance of the WordCount application using 16 virtual machines.

DOI: 10.1002/cpe

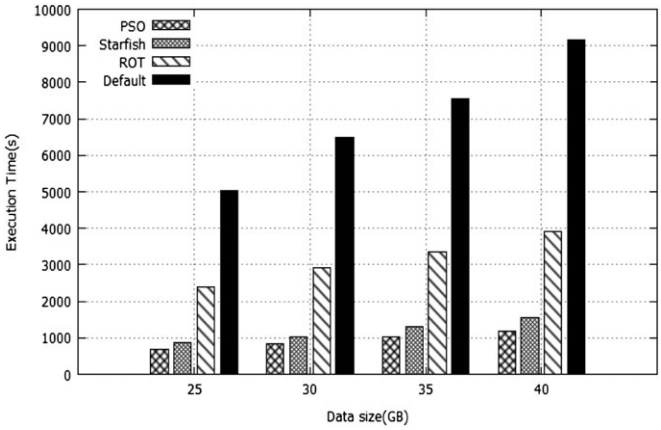

Figure 12. The performance of the Sort application using 16 virtual machines.

6. RELATED WORK

In recent years, numerous researches have been carried out to optimize the performance of Hadoop from different aspects. The methodologies of these studies are diverse and range from optimizing Hadoop job scheduling mechanisms to tuning the configuration parameter settings. For example, many researchers have focused on developing adaptive load balancing mechanisms [38–41] and data locality algorithms [42–45] to improve the performance of Hadoop.

A group of researchers have proposed optimization approaches for a particular type of jobs such as short jobs and query based jobs [46–49]. Jahani *et al*. proposed the MANIMAL model [46], which automatically analyzes a Hadoop program using a static analyzer tool for optimization. However, the MANIMAL model only focuses on relational style programs employing the selection and projection operators and does not consider text-processing programs. Moreover, it only optimizes the map phase in Hadoop. Elmeleegy *et al*. presented Piranha [49], a system which optimizes short jobs (i.e., query base jobs) by minimizing their response times. They suggested that fault-tolerance facilities are not necessary for short running jobs because the jobs are small, and they are unlikely to incur failures. The works presented in [47, 48] focus on optimizing short Hadoop jobs by enhancing tasks execution mechanisms. They optimized task initialization and termination stages by removing the constant heartbeat, which is used for the tasks set up and clean up process in Hadoop. They proposed a push-model for heartbeat communication to reduce delays between the JobTracker and a TaksTracker, and implemented an instance communication mechanism between the JobTraker and a TaskTracker in order to separate message communication from the heartbeat.

Many researchers have also researched into resources provisioning for Hadoop jobs. Palanisamy *et al*. presented the Cura model [50] that allocates an optimum number of VMs to a user job. The model dynamically creates and destroys the VMs based on the user workload in order to minimize the overall cost of the VMs. Virajith *et al*. [51] proposed Bazaar that predicts Hadoop job performance and provisions the resources in term of VMs to satisfy user requirements. A model proposed in [52] optimizes Hadoop resource provisioning in the Cloud. The model employed a brute-force search to find optimum values for map slots and reduce slots over the resource configuration space. Tian *et al*. [53] proposed a cost model that estimates the performance of a Hadoop job and provisions the resources for the job using a simple regression technique. Chen *et al*. [54] further improved the cost model and proposed CRESP, which employs a brute-force search technique for provisioning optimal resources in term of map slots and reduce slots for Hadoop jobs. Lama *et al*. [55] proposed AROMA, a system that automatically provisions the optimal resources of a job to achieve service level objectives. AROMA builds on a clustering technique to group the jobs with similar behaviors. It employed support vector machine to predict the performance of a Hadoop job and a pattern search technique to find an optimal set of resources for a job to achieve the required deadline with a minimum cost. However, AROMA cannot predict the performance of a job

DOI: 10.1002/cpe

whose resource utilization pattern is different from any previous ones. More importantly, AROMA does not provide a comprehensive mathematical model to estimate a job execution time.

There are a few other sophisticated models such as [12, 13, 15–17] that are similar to the proposed work in the sense that they optimize a Hadoop job by tuning the configuration parameter settings. Wu *et al*. proposed PPABS [15], which automatically tunes the Hadoop framework configuration parameter settings based on executed job profiles. The PPABS framework consists of analyzer and recognizer components. The analyzer trains the PPABS to classify the jobs having similar performance into a set of equivalent classes. The analyzer uses *K-means++* to classify the jobs and simulated annealing to find optimal settings. The recognizer classifies a new job into one of these equivalent classes using a pattern recognition technique. The Recognizer first runs the new job on a small dataset using default configuration settings and then applies the pattern recognition technique to classify it. Each class has the best configuration parameter settings. Once the recognizer determines the class of a new job then it automatically uploads the best configuration settings for this job. However, PPABS is unable to find the fine-tuned configuration settings for a new job, which does not belong to any of these equivalent classes. Moreover, PPABS does not consider the correlations among the configuration parameters. Herodotou *et al*. proposed Starfish [12, 13] that employs a mixture of cost model [14] and simulator to optimize a Hadoop job based on previously executed job profile information. Starfish divides the search space into subspaces. It considers the configuration parameters independently for optimization and combines the optimum configuration settings found in each subspace as a group of optimum configuration settings. Starfish collects the running job profile information at a fine-granularity for job estimation and automatic optimization. However, collecting detailed job profile information with a large set of metrics generates an extra overhead. As a result, the Starfish model is unable to accurately estimate the job execution time due to which it overestimates the values for some configuration parameters especially for the number of reduce tasks. As Starfish divides the configuration parameter space into subspaces which may ignore the correlations among the parameters. Liao *et al*. proposed Gunther [16], a search based model that automatically tunes the configuration parameters using genetic algorithm. One critical limitation of Gunther is that it does not have a fitness function in the implemented genetic algorithm. The fitness of a set of parameter values is evaluated through physically running, a Hadoop job using these parameters which is a time consuming process. Liu *et al*. [17] proposed Panacea with two approaches to optimizing Hadoop applications. In the first approach, it optimizes the compiler at run time, and a new API was developed on top of Soot [56] to reduce the overhead of iterative Hadoop applications. In the second approach, it optimizes a Hadoop application by tuning Hadoop configuration parameters. In this approach, it divides the parameters search space into sub-search spaces and then searches for optimum values by trying different values for parameters iteratively within the range. However, Panacea is unable to provide a sophisticated search technique and a mathematical function, which represents a correlation of the Hadoop configuration parameters. Li *et al*. [18] proposed a performance evaluation model for the whole system optimization of Hadoop. The model analyzes the hardware and software levels and explores the performance issues in these layers. The model mainly focuses on the impact of different configuration settings on a job performance instead of tuning the configuration parameters.

7. CONCLUSION

Running a Hadoop job on default parameter settings has led to performance issues. This paper optimized Hadoop performance by automatically tuning its configuration parameters. The proposed work employed GEP to build up an objective function that represents a correlation among the Hadoop configuration parameters and implemented PSO to further search for optimal or near optimal parameter settings. The proposed work improves Hadoop performance significantly in comparison with the default settings. In addition, the proposed work performs better than existing representative works such as the ROT work and the Starfish model in Hadoop performance optimization.

DOI: 10.1002/cpe

We have implemented some work on Hadoop job estimation and resource provisioning [21]. One future work will be to integrate the two works together for resource provisioning in optimized Hadoop clusters.

ACKNOWLEDGEMENTS

This research is supported by the UK EPSRC under grant EP/K006487/1 and also the National Basic Research Program (973) of China under grant 2014CB340404.

REFERENCES

1. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004; 10.
2. Apache Hadoop. *Apache*. [Online]. Available: http://hadoop.apache.org/. [Accessed: 18-Feb-2015].
3. Khan M, Ashton PM, Li M, Taylor GA, Pisica I, Liu J. Parallel detrended fluctuation analysis for fast event detection on massive PMU data. *Smart Grid, IEEE Transactions* 2015; **6**(1):360–368.
4. Khan M, Li M, Ashton P, Taylor G, Liu J. Big data analytics on PMU measurements. In *Fuzzy Systems and Knowledge Discovery* (*FSKD*), *2014 11th International Conference on*, 2014; 715–719.
5. Kang U, Tsourakakis CE, Faloutsos C. PEGASUS: Mining peta-scale graphs. *Knowledge and Information Systems* 2011; **27**(2):303–325.
6. Panda B, Herbach JS, Basu S, Bayardo RJ. PLANET: Massively parallel learning of tree ensembles with MapReduce. *Proceedings of the VLDB Endowment* 2009; **2**(2):1426–1437.
7. Pavlo A, Paulson E, Rasin A. A comparison of approaches to large-scale data analysis. In *SIGMOD '09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009; 165–178.
8. Babu S. Towards automatic optimization of MapReduce programs. *Proc. 1st ACM Symp. Cloud Comput. - SoCC '10*, 2010; 137.
9. Hadoop Performance Tuning. [Online]. Available: https://hadoop-toolkit.googlecode.com/files/White paper-HadoopPerformanceTuning.pdf. [Accessed: 23-Feb-2015].
10. 7 tips for Improving MapReduce Performance. 2009. [Online]. Available: http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/. [Accessed: 20-Feb-2015].
11. White T. *Hadoop:The Definitive Guide* (3rd edn). Yahoo press: Sebastopol, CA, USA, 2012; 688.
12. Herodotou H, Babu S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *PVLDB* 2011; **4**(11):1111–1122.
13. Herodotou H, Lim H, Luo G, Borisov N, Dong L, Cetin FB, Babu S. Starfish: a self-tuning system for big data analytics. In *CIDR*, 2011, 261–272.
14. Herodotou H. Hadoop Performance Models. 2011. [Online]. Available: http://www.cs.duke.edu/starfish/files/hadoop-models.pdf. [Accessed: 22-Oct-2013].
15. Wu D, Gokhale AS. A self-tuning system based on application profiling and performance analysis for optimizing Hadoop MapReduce cluster configuration. In *20th Annual International Conference on High Performance Computing*, *HiPC 2013*, *Bengaluru* (*Bangalore*), *Karnataka*, *India*, *December 18–21*, *2013*, 2013; 89–98.
16. Liao G, Datta K, Willke TL. Gunther: search-based auto-tuning of Mapreduce. In *Proceedings of the 19th International Conference on Parallel Processing*, 2013; 406–419.
17. Liu J, Ravi N, Chakradhar S, Kandemir M. Panacea: towards holistic optimization of MapReduce applications. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, 2012; 33–43.
18. Li Y, Wang K, Guo Q, Li X, Zhang X, Chen G, Liu T, Li J. Breaking the boundary for whole-system performance optimization of big data. In *Low Power Electronics and Design* (*ISLPED*), *2013 IEEE International Symposium on*, 2013; 126–131.
19. Kennedy J, Eberhart R. Particle swarm optimization. In *Proceedings., IEEE International Conference on Neural Networks 1995*. 1995; 4:1942–1948.
20. Eberhart R, Kennedy J. A new optimizer using particle swarm theory. In *Micro Machine and Human Science*, *1995. MHS '95.*, *Proceedings of the Sixth International Symposium on*, 1995; 39–43.
21. Khan M, Jin Y, Li M, Xiang Y, Jiang C. Hadoop performance modeling for job estimation and resource provisioning. *Parallel Distrib. Syst. IEEE Trans.* 2015; **PP**(99):1. PrePrints, doi: 10.1109/TPDS.2015.2405552.
22. Ferreira C. Gene expression programming: a new adaptive algorithm for solving probles. *Complex Systems* 2001; **13**(2):87–129.
23. Bäck T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press: Oxford, UK, 1996.
24. Holland JH. *Adaptation in Natural and Artificial Systems*. MIT Press: Cambridge, MA, USA, 1992.
25. Koza JR. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press: Cambridge, MA, USA, 1992.
26. Kennedy J, Eberhart RC, Shi Y. *Swarm Intelligence*. Morgan Kauffmann Publishers: San Francisco, California, 2001; 369–392.
27. Parsopoulos KE, Vrahatis MN. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing* 2002; **1**(2–3):235–306.

28. del Valle Y, Venayagamoorthy GK, Mohagheghi S, Hernandez JC, Harley RG. Particle swarm optimization: basic concepts, variants and applications in power systems. *Evolutionary Computation, IEEE Transactions* 2008; **12**(2):171–195.
29. Sadasivam GS, Selvaraj D. A novel parallel hybrid PSO-GA using MapReduce to schedule jobs in Hadoop data grids. In *Nature and Biologically Inspired Computing* (*NaBIC*), *2010 Second World Congress on*, 2010; 377–382.
30. Ludwig S. Scalability analysis: reconfiguration of overlay networks using nature-inspired algorithms. In *Advances in Intelligent Modelling and Simulation*, Kołodziej J, Khan SU, Burczy´nski T (eds), vol. **422**, Springer: Berlin Heidelberg, 2012; 137–154.
31. Engelbrecht A. Particle swarm optimization: Velocity initialization. In *Evolutionary Computation* (*CEC*), *2012 IEEE Congress on*, 2012; 1–8.
32. Jiao B, Lian Z, Gu X. A dynamic inertia weight particle swarm optimization algorithm. *Chaos, Solitons & Fractals* 2008; **37**(3):698–705.
33. Helwig S, Wanka R. Particle swarm optimization in high-dimensional bounded search spaces. In *Swarm Intelligence Symposium*, *2007. SIS 2007. IEEE*, 2007; 198–205.
34. Helwig S, Branke J, Mostaghim SM. Experimental analysis of bound handling techniques in particle swarm optimization. *Evolutionary Computation, IEEE Transactions* 2013; **17**(2):259–271.
35. Li-ping Z, Huan-jun Y, Shang-xu H. Optimal choice of parameters for particle swarm optimization. *Journal of Zhejiang University Science* 2005; **6**(6):528–534.
36. Ai-min L, Zhang H, Yang G, Lin X. Research and application of multi-objective particle swarm optimization in linear induction motor operating mechanism. In *Electric Power Equipment - Switching Technology* (*ICEPE-ST*), *2011 1st International Conference on*, 2011; 291–295.
37. McCaffrey J. Artificial intelligence: particle swarm optimization. *Microsoft-MSDN Magazine* 2011.
38. Mao H, Hu S, Zhang Z, Xiao L, Ruan L. A Load-driven task scheduler with adaptive DSC for MapReduce. In *Green Computing and Communications* (*GreenCom*), *2011 IEEE/ACM International Conference on*, 2011; 28–33.
39. Nanduri R, Maheshwari N, Reddyraja A, Varma V. Job aware scheduling algorithm for MapReduce framework. In *Cloud Computing Technology and Science* (*CloudCom*), *2011 IEEE Third International Conference on*, 2011; 724–729.
40. Vernica R, Balmin A, Beyer KS, Ercegovac V. Adaptive MapReduce using situation-aware mappers. In *Proceedings of the 15th International Conference on Extending Database Technology*, 2012; 420–431.
41. You HH, Yang CC, Huang JL. A load-aware scheduler for MapReduce framework in heterogeneous cloud environments. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011; 127–132.
42. Hammoud M, Sakr MF. Locality-aware reduce task scheduling for MapReduce. In *Cloud Computing Technology and Science* (*CloudCom*), *2011 IEEE Third International Conference on*, 2011; 570–576.
43. He C, Lu Y, Swanson D. Matchmaking: a new MapReduce scheduling technique. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011; 40–47.
44. Xie J, Yin S, Ruan X, Ding Z, Tian Y, Majors J, Manzanares A, Qin X. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *IPDPS Workshops*, 2010; 1–9.
45. Heintz B, Chandra A, Sitaraman RK. Optimizing MapReduce for Highly Distributed Environments. 2012.
46. Jahani E, Cafarella MJ, Ré C. Automatic optimization for MapReduce programs. *Proceedings of the VLDB Endowment* 2011; **4**(6):385–396.
47. Yan J, Yang X, Gu R, Yuan C, Huang Y. Performance optimization for short MapReduce job execution in Hadoop. In *Cloud and Green Computing* (*CGC*), *2012 Second International Conference on*, 2012; 688–694.
48. Gu R, Yang X, Yan J, Sun Y, Wang B, Yuan C, Huang Y. SHadoop: improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters. *Journal of Parallel and Distributed Computing* 2014; **74**(3):2166–2179.
49. Elmeleegy K. Piranha: optimizing short jobs in Hadoop. *Proceedings of the VLDB Endowment* 2013; **6**(11):985–996.
50. Palanisamy B, Singh A, Langston B. Cura: A Cost-optimized model for MapReduce in a cloud. In *Parallel Distributed Processing* (*IPDPS*), *2013 IEEE 27th International Symposium on*, 2013; 1275–1286.
51. Virajith J, Hitesh B, Paolo C, Thomas K, Antony R. Bazaar: enabling predictable performance in datacenters. 2012.
52. Kambatla K, Pathak A, Pucha H. Towards optimizing Hadoop provisioning in the cloud. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, 2009.
53. Tian F, Chen K. Towards optimal resource provisioning for running MapReduce programs in public clouds. In *2011 IEEE 4th International Conference on Cloud Computing*, 2011; 155–162.
54. Chen K, Powers J, Guo S, Tian F. CRESP: towards optimal resource provisioning for MapReduce computing in public clouds. *IEEE Transactions on Parallel and Distributed Systems* 2014; **25**(6):1403–1412.
55. Lama P, Zhou X. AROMA: automated resource allocation and configuration of Mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*, 2012; 63–72.
56. Vallée-Rai R, Co P, Gagnon E, Hendren L, Lam P, Sundaresan V. Soot - a Java Bytecode optimization framework. In *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research*, 1999; 13.

DOI: 10.1002/cpe